**Task1\_19101100**

input\_graph=[]

with open ('/content/sample\_data/Question1\_input 2.txt') as input1\_a1:

  file1=input1\_a1.readlines()

for line in file1:

  l=line.split()

  input\_graph.append(l)

# print(input\_graph)

x\_dimentions=[0,1]                                                                                                          #defining steps we need to move to horizontally/diagonally

y\_dimentions=[1,0,-1]                                                                                                       #defining steps we need to move to vertically/diagonally

#dimentions=[[0,1],[0,-1],[1,1],[1,0],[1,-1]]

max\_region\_count=[]                                                                                                         #stores count of no of infected people in all regions

def find\_max\_infected\_region(row, column,count\_infected):                                                                   #finds max no of people infected

  # print("current row and column:",row,column)

  # print("node then:",input\_graph[row][column])

  if column<=len(input\_graph[0])-1 and row<=len(input\_graph)-1 and column>=0:                                               #limit set so that the 'Y' we find is not out of the range of the matrix

    if input\_graph[row][column] == "Y":

      input\_graph[row][column]="Visited"                                                                                    #turn Y to Visited if index visited.

      # print("count now:",count\_infected)

  if count\_infected>=6:

    count\_infected+=1

  for step1 in x\_dimentions:

    for step2 in y\_dimentions:

      if column+step2<=len(input\_graph[0])-1 and row+step1<=len(input\_graph)-1 and column+step2>=0 and row+step1>=0:        #limit set so that next step we move to is not out of the range of the matrix

        if(input\_graph[row+step1][column+step2]!=input\_graph[row][column] and input\_graph[row+step1][column+step2]=="Y"):   #makes sure the next index has Y and that it is not calculating for the current index where we got Y

          count\_infected+=1                                                                                                 #infected++ if in next travelled index, Y is found.

          find\_max\_infected\_region(row+step1,column+step2,count\_infected)                                                   #recurssive call to find Y for the next indexes to traverse from current Y

  max\_region\_count.append(count\_infected)                                                                                   #total number of Y in a region found. So number added to list

  # print(count\_infected)

count=0

for row in range(len(input\_graph)):

  for col in range(len(input\_graph[0])):

    # print(input\_graph[row][col],end=",")

    if(input\_graph[row][col]=="Y"):

      find\_max\_infected\_region(row,col,count+1)                                                                             #after an index with Y found, calls find\_max\_infected\_region function to find max infected region

# print(max\_region\_count)

print(max(max\_region\_count))                                                                                                #max value from the list of number of infected people in all regions printed

**Task2\_19101100**

xcity\_grid=[]

with open ('/content/sample\_data/Question2 input1.txt') as input2\_a1:

  row\_num=input2\_a1.readline()

  column\_num=input2\_a1.readline()

  file1=input2\_a1.readlines()

# print(row\_num)

# print(column\_num)

for line in file1:

  l=line.split()

  xcity\_grid.append(l)

# print(xcity\_grid)

alien\_position=[]

time=0

def find\_humans():                                                                                                     #function to find total number of humans initially

  humans\_num=0

  for row in range(len(xcity\_grid)):

      for column in range(len(xcity\_grid[0])):

        if xcity\_grid[row][column]=='H':

          humans\_num+=1

  return humans\_num

def find\_alien(humans, xcity\_grid,step):

    # print("humans",humans)

    for row in range(len(xcity\_grid)):

      for column in range(len(xcity\_grid[0])):

        if xcity\_grid[row][column]=='A':

          list\_position=[row,column]                                                                                    #calculates position of alien when it is found

          if list\_position not in alien\_position:                                                                       #if newly calculated aliens position already not in list of all aliens position, added to list

            alien\_position.append(list\_position)

    if step<4:

        alien\_attack(humans,alien\_position,step)                                                                        #alien attack called to turn surrounding horizontal/vertical humans to aliens

    else:

        print("Time",step,"minutes")

        if humans==0:

          print("No one survived")

        else:

          print(f'{num\_humans-(num\_humans-humans)} survived')                                                           #surviving number of humans calculated. num\_humans=total humans initially. num\_humans-humans=humans who turned aliens.

        pass                                                                                                            #so remaining humans=total humans-humans who turned into aliens. pass when result found

# print(alien\_position)

x\_dimentions=[0,1,-1]                                                                                                   #defining steps we need to move to horizontally

y\_dimentions=[1,0,-1]                                                                                                   #defining steps we need to move to vertically

def alien\_attack(humans,aliens,step):

  # print(humans)

  # print(aliens)

  for x in range(len(aliens)):

    row=aliens[x][0]                                                                                                    #first index of each list in aliens i.e. alien\_position list=row no of current alien

    column=aliens[x][1]                                                                                                 #second index of each list in aliens i.e. alien\_position list=column no of current alien

    for step1 in x\_dimentions:

      for step2 in y\_dimentions:

        if (step1==0 and step2==1) or (step1==0 and step2==-1) or (step1==1 and step2==0) or (step1==-1 and step2==0):  #moves only when grid is vertical or horizontal so condition given to ensure that

          if column+step2<=len(xcity\_grid[0])-1 and row+step1<=len(xcity\_grid)-1 and column+step2>=0 and row+step1>=0:  #limit set so that next step we move to is not out of the range of the matrix

            if(xcity\_grid[row+step1][column+step2]=="H"):

              xcity\_grid[row+step1][column+step2]="A"                                                                   #human turned to alien if found horizontally or vertically to an alien

              humans-=1                                                                                                 #one less human each time human turns to alien

              # print(humans)

  # print(xcity\_grid)

  step+=1                                                                                                               #increased level/step/time after each attack by aliens

  # print(step)

  find\_alien(humans,xcity\_grid,step)                                                                                    #recurssion call to find positions of all current aliens again

num\_humans=find\_humans()                                                                                                #stores total number of humans

# print(num\_humans)

find\_alien(num\_humans, xcity\_grid,time)                                                                                 #calls function to find positions of all current aliens

# print(xcity\_grid)